Evaluating Performance of Android Systems as a Platform for Augmented Reality Applications

Andrés L. Sarmiento, Margarita Amor, Emilio J. Padrón, Carlos V. Regueiro, Raquel Concheiro, and Pablo Quintía
Dept. Electronics and Systems
University of A Coruña
A Coruña, Spain
andreslopezsarmiento@gmail.com, margarita.amor@udc.es, emilioj@udc.es, cvazquez@udc.es
rconcheiro@udc.es, pquintia@udc.es

***Abstract*—Android is a free operating system for mobile devices that has become very popular these days. In this work we analyse the capabilities of an Android smartphone with the OpenGL ES API for the rendering of synthetic realistic images. The aim is to find out the facilities and the main limitations of the platform for the development of augmented reality games, studying the integration of synthetic information with the real environment using data from the camera and the positioning engine. Thus, our research covers mainly three fields: an analysis of the information provided by the camera, a study of the tracking and positioning capabilities of current Android devices and an outline of the rendering facilities usually found in these devices. The performance, in terms of frames per second and latency, has been tested in different smartphones, in addition to evaluate the reliability and efficiency of the sensors and the quality of rendering. In order to show all the results obtained from this study we have developed an augmented reality game trying to combine quality, performance and real-time interaction.**

***Keywords*-Augmented reality; Android; Positioning sensors; Image processing; Realistic image synthesis**

I. Introduction

This work extends a previous survey [1] about the current state and capabilities of augmented reality applications in Android smartphones, adding new tests and devices to the analysis. Smartphones have gathered functionalities and features of an increasingly number of different devices, from those used in a more professional environment (i.e., mobile devices, electronic agendas, GPS) to others with recreational aspects (such as cameras or video game consoles). Although this means an obvious saving of money and space, the major advantage of these new devices is the integration of all those capabilities in increasingly complex and innovative applications.

Most of the operating systems available for these devices have been developed ad hoc for each model, such as Apple's *iPhone OS* [2] or Samsung's *Bada* [3]. Android [4], however, has a very different origin since it is a multi-platform linux-based OS (rigorously, Android is really a software stack that includes an OS, middleware and a handful of applications) promoted by a group of companies. This open source and cross-platform nature, together with the growth it has experienced over the past few years, giving access to a wide range of devices ranging from the low price terminals to the more expensive ones, made us adopt Android as the platform for this work.

Augmented reality (AR) [5] is one of the newest and most popular applications that have recently shown up within the sphere of smartphones. Most of the existing proposals may be classified at one of the following three groups: AR browsers; applications that allow us to move through a completely synthetic environment; and, lastly, applications that use the camera information to show virtual objects in the phone.

AR browsers are outdoor augmented reality applications that do geopositioning of virtual objects in the real world by using the orientation sensors and the GPS or a triangulation positioning system to determine the position where they must be placed [6], [7]. The information about the objects to be positioned is pre-computed and these applications do not demand a great accuracy in the positioning and orientation of the mobile device. AR browsers generally presents a good operation, showing real time information with an acceptable precision though with the typical limitations of any GPS or triangulation positioning systems. Generally speaking, the positioning accuracy is upper than 5 meters in optimal conditions (open space and good firmament visibility). However, the precision drops drastically in cities and, above all, inside buildings.

The second type of AR applications uses only the movement and orientation of the device to readjust the vision of a synthetically generated scene [8], [9]. In these applications all elements are generated in a virtual scene that is shown in the mobile screen. The device movements and its orientation in relation to the Earth's magnetic field and to the centre of the Earth (gravity) are used to establish or update the point of view in the scene shown in the screen. The image captured by the camera can also be shown, but it has not any influence in the applications as it is not really processed by the device.

Finally, some applications apply artificial vision techniques [10], [11]. This type of applications processes the perceived image and uses that information to put the virtual models in the right place, usually through known tags to be able to interpret in a better way the perceived information. Obviously, this approach means higher computational requirements and a greater application complexity. As a consequence, there are really few AR applications in Android based on exploiting data obtained by the camera and most of them are basically technical demonstrations. On the other hand, this kind of applications can be found in other systems apart from mobile devices, such as desktop computers, mainly due to the usual high computing requirements of image processing algorithms.

To sum up, from a user point of view there are multiple applications using geopositioning of objects with different objectives, and several proposals already in the market to cover this topics. However, few Android applications use artificial vision to perceive and process the real world, and those have usually a poor performance. In our research, we focus on this last line of work since the best approach to integrate synthetic information with the immediate real-time data from the environment in a realistic scenario such as a dynamic and complex environment seems to be the exploitation of both the camera and the positioning sensors of these devices. Since Android is a brand new platform, analysing the viability of this kind of AR application is a necessary preliminary step. This analysis is complemented in this work with the development of a simple AR game for indoor environments as a demonstration of the possibilities of this approach.

The structure of the paper is as follows, Section II goes into the study of Android smartphones as an AR platform. We have divided our analysis in three big sections: firstly, a study of the possibilities for processing the information captured by the camera; next, a survey of the positioning and tracking capabilities of these smartphones in an indoor environment and, lastly, the possibilities for the real-time rendering of complex virtual models. A brief outline of all the aspects studied in the analysis is in the end of this section. Section III describes the AR game we have developed taking into account the results from our analysis, and Section IV shows the performance achieved with our proposal. Finally, the conclusions we have reached with this work are shown.

II. Analysis of the capabilities of an Android smartphone with OpenGL ES

In this section, an analysis of the capabilities of the Android platform in the context of AR is presented. Table I shows the main features of the devices used in our study. These devices are representative of the current smartphone market in the last couple of years.

A. *Image capture and processing*

The camera of a smartphone is of great importance for AR applications, since the synthetic images are usually rendered over real images obtained by the camera. If the image from the camera is just being displayed, Android efficiently add it to the rest of layers shown by the application. Otherwise, if the image is going to be processed, it is captured by the system and provided to the application as a vector of bytes in a default format previously set in the camera. Many cameras (such as the ones used in our analysis) only work with the YUV image format.

Once an image from the camera is obtained, any image processing technique may be applied on it. Since image processing is usually a high-cost computationally task, any operation has to be spawned in a different thread to the one running the application's GUI. Otherwise, non-responding lags are probably to be experienced in the application. Besides, it is also a good practice to code image processing tasks in native code (C, C++) and use the NDK to integrate it in the application [12]. This way, we can achieve an important improvement, up to 400%, in the velocity of execution.

In order to analyse the possibilities of image capture and processing at iterative rates we started studying the maximum frequency at which data can be obtained. This allow us to get the top level of performance that can be achieved in these devices. Thus, our first test just captures an input image and calls a naive processing image code that just computes the frame rate (fps, frames per second) with no additional computation (the input image is not processed at all). The result obtained with this simple test for a *Motorola Milestone* with Android v2.1 and a configuration of $10\,\mathrm{fps}$ as the maximum capture frequency was $8.8\,\mathrm{fps}$, whereas a maximum of $9.3\,\mathrm{fps}$ was obtained when the maximum frequency was set to $30\,\mathrm{fps}$. Obviously, these results are far from being satisfactory, since even without any image processing we are rounding the minimum frame rate acceptable for a fluid interaction.

To study the effect of a simple image processing on the performance, we have extended our test by adding the display on the screen of the images obtained by the camera. Since images are obtained from the camera in YUV format for portability issues and they must be in RGB to be displayed by Android, some computation is needed to get the conversion. Therefore, this test program just takes each image captured by the camera, recodes it from YUV to RGB and gets it displayed on the screen. Additionally, our test program can be configured to encode only a region of the image. The results of running our tests in the *Motorola Milestone* are depicted in Table II. The table shows the frame rate as a function of the size of the region to process and the highest frequency set in the camera. As can be observed, a top value of $5.15\,\mathrm{fps}$ has been obtained, that does not make

Table I: Technical data for the smartphones used in our tests.

Android	*Motorola Milestone* 2.1 Eclair	*GeeksPhone One* 2.2 Froyo	*Samsung Galaxy S* 2.2 Froyo	*HTC Sensation* 4.0 ICS	*Samsung Galaxy S2* 2.3 Gingerbread / 4.0 ICS
CPU	ARM Cortex A8 550 MHz	ARM11 528 MHz	Samsung Hummingbird 1 GHz	Qualcomm Scorpion dual-core 1.2 GHz	ARM Cortex A9 dual-core 1.2 GHz
GPU	PowerVR SGX 530	built-in	PowerVR SGX 540	Qualcomm Adreno 220	ARM Mali-400 MP
RAM	256 MB	256 MB	512 MB	768 MB	1 GB
Display	3.7" 854x480	3.2" 400x240	4" 800x480	4.3" 960x540	4.3" 800x480
GPS	✓	✓	✓	✓	✓
Acceler.	✓	✓	✓	✓	✓
Compass	✓	✗	✓	✓	✓
Camera	✓	✓	✓	✓	✓

possible to keep a fluid stream of images on the screen. Furthermore, we have observed a delay of about one second in what is being displayed. A further analysis of this delay is presented at the end of this section.

These results show that the configuration with 10 fps as the maximum frequency obtains the best results; probably because with this frequency the application is not saturated with images it is not able to process. Even though there is a substantial improvement in the capture by reducing the image size (25% fps with $1/4$ of the size), this results in less than 5 fps with the max set to 30 fps.

In a next step we have studied the performance of image processing, so a simple colour segmentation of pixels is carried out. Since all the pixels in the image were already being processed by the image recoding process in the previous test, adding colour segmentation only needs a few additional lines of code, so execution times remains almost the same, as we have tested experimentally. Other tests adding different image processing algorithms were carried out and similar execution times were obtained.

The obvious conclusion coming from the results of our tests is that the image processing velocity is really low in Android v2.1 and previous versions, obtaining a slow response even after implementing optimisations such as using NDK and running the processing in a different thread. The main reason for this performance seems to be in the process the system follows for each image captured by the camera, allocating memory, saving a copy of the image, calling the function to process it and, finally, removing the reference to the allocated memory [13]. This whole process entails a completely inefficient memory management, that is made still more acute by the high cost of garbage collection in Android, between 100 and 300 milliseconds. Not reusing the memory assigned to each image results in a frequent invocation of the garbage collector, burdening the performance.

This important issue with memory management was solved in Android v2.2, that included other significant improvements as well, such as a *Just in Time* compiler. Regarding image processing, the API was also enhanced with new methods that work with a buffer passed as a parameter, removing the memory allocation and removal for each image to process.

Table II: Image capture, decoding and visualisation on *Motorola Milestone* with Android v2.1.

Image size	*Max. FPS* *Milestone*-30	*Milestone*-10
560×320	3.25	3.90
280×320	3.90	4.45
280×160	4.50	4.95
140×160	4.60	5.10
15×15	4.65	5.15

Table III: Image capture, decoding and visualisation in devices with Android v2.2.

GeeksPhone Size	FPS	*Galaxy S* Size	FPS
400×240	3.90	800×480	5.70
200×240	4.50	400×480	7.10
200×120	5.00	400×240	8.00
100×120	5.50	200×240	8.75
15×15	5.80	15×15	9.20

We have analysed the improvements in Android v2.2 by running the same tests in two of our devices with this version of the OS. Table III shows the results obtained with Android v2.2 for the simple capture and recoding test previously outlined in Table II, in this case considering only the configuration of 10 fps as the maximum frequency, since it provides the best results and the 30 fps configuration does not add relevant information to the analysis. As can be observed, there is a performance increase of 50%, from 3.90 up to 5.70, and taking into account a 50% increase in the image size as well. The improvement is even more appreciable looking at the visualisation delay, that has been reduced from around 1 second to 0.5 seconds.

Table IV depicts the results achieved by the new Android v4.0 running on two current smartphones: HTC Sensation and Samsung Galaxy S2. As can be observed, the performance has significantly improved, above all for the 30 fps configuration, that now even achieves the best results in some of the cases. The frame rates shown in this table

Table IV: Image capture, decoding and visualisation on *HTC Sensation* and *Galaxy S2* with Android v4.0.

	Max. FPS			
Image size	*Sensation*-30	*Sensation*-10	*Galaxy S2*-30	*Galaxy S2*-10
640×480	9.38	9.62	11.89	11.59
320×480	10.36	10.50	11.93	11.87
320×240	12.20	12.20	12.08	12.04
160×240	12.64	12.71	13.26	12.06
160×120	13.28	13.05	14.87	12.63

Table V: Image capture, decoding and visualisation in a high load scenario on *HTC Sensation* and *Galaxy S2* with Android v4.0.

	Max. FPS			
Image size	*Sensation*-30	*Sensation*-10	*Galaxy S2*-30	*Galaxy S2*-10
640×480	7.60	7.70	8.60	8.73
320×480	8.00	8.10	9.41	9.23
320×240	8.40	8.50	12.03	9.60
160×240	8.90	8.70	13.03	9.76
160×120	9.10	8.90	14.42	10.01

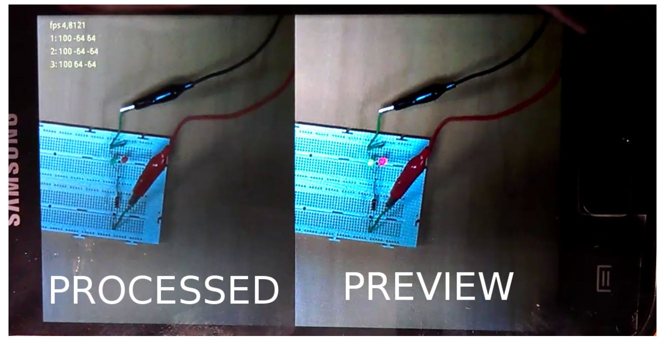
were obtained while keeping a small amount of background workload in the smartphone, as in the previous tests. In order to check how the background processes running in the smartphone influence the performance of the image capture and process task, we have repeated the test in the Android v4.0 devices while a great bunch of usual applications were being executed in background (IM, e-mail, alerts. . .). The results in such a scenario are shown in Table V, and an important drop in performance can be observed compared to Table IV, above all in the *HTC Sensation*, even though both two devices are dual core.

Lastly, all our tests analysing the image capture and processing in Android have revealed an important delay in the capture of the input data. This delay was extremely high in the devices with Android v2.1, about 1 second, and has been reduced in the last versions. Table VI shows the results of a simple experiment to measure this delay in three different devices, *Samsung Galaxy S*, with Android v2.2, and *HTC Sensation* and *Samsung Galaxy S2*, with Android v4.0. Our experiment involved the measurement of the response time of a simple quantifiable event, the off/on switching of a LED, that allow us to know the delay in the image capture for each device (see Figure 1). As can be observed, the delay in the *Samsung Galaxy S2* is about half the time it is in the other two devices, even though one of them, *HTC Sensation*, is using the same version of Android. A video with this experiment is provided as additional material with this paper.

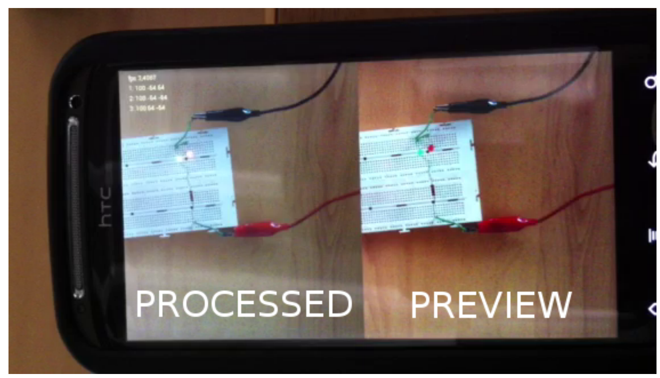
Summing up, although the improvements introduced with Android v2.2 and next versions make us optimistic about future revisions, above all in combination with more powerful hardware such as the recent multi core processors,

Table VI: Delay between image capture and processing on *Galaxy S* with Android v2.2 and *HTC Sensation* and *Galaxy S2* with Android v4.0.

Delay (s)	*Galaxy S*	*Sensation*	*Galaxy S2*
Average	0.454	0.414	0.246
Std. deviation	0.024	0.041	0.060



(a)



(b)

Figure 1: Delay time measurements in (a) *Samsung Galaxy S* and (b) *HTC Sensation*.

the current situation does not allow real time applications entirely based on the processing of images from the camera. Thus, an efficiency analysis of the real world around us makes necessary the use of data from other sources, e.g., positioning sensors.

B. Device positioning and orientation

In this subsection we outline the main positioning and tracking sensors included in most Android smartphones: accelerometer, compass and GPS. In order to check their performance, some test were executed on our *Milestone* phone, similar results were obtained in the rest of devices.

An accelerometer measures the proper acceleration of itself, i.e., a change in velocity, that involves a change in position. Mathematically velocity is the integral of acceleration, and position is the integral of velocity. Smartphones have usually three accelerometers, one for each spatial axis.

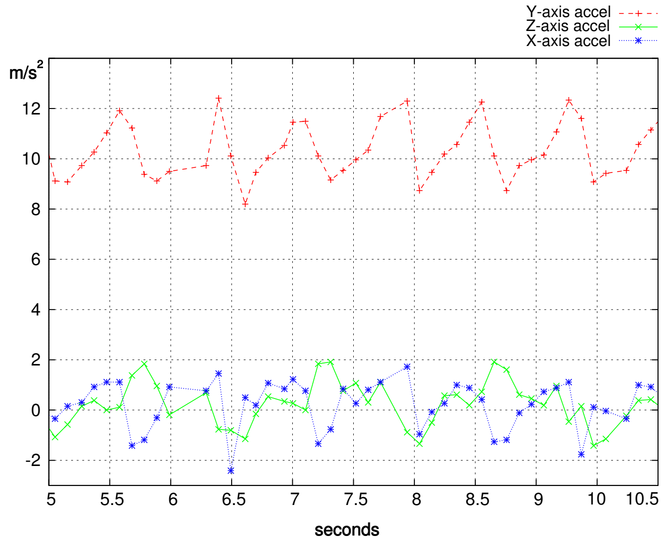


Figure 2: Values obtained by the accelerometers of a *Motorola Milestone* during a user's walk.

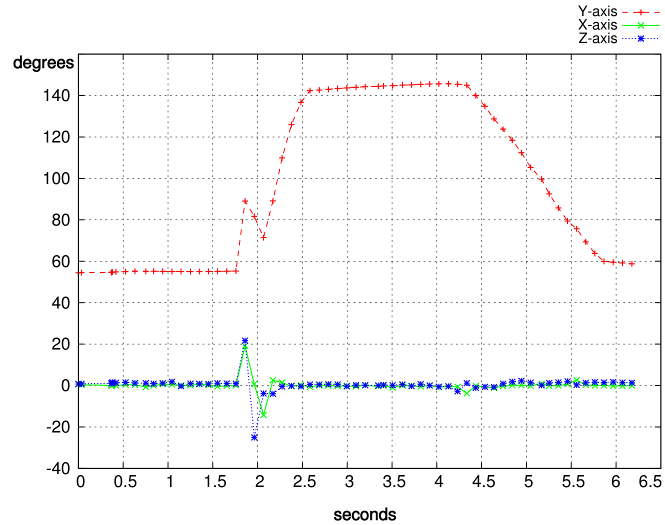


Figure 3: Values obtained by the compasses of a *Motorola Milestone* with a 90° turn.

Theoretically, the position of a smartphone could be guessed from data provided by these sensors. Data are presented as a vector with the values measured for each of the three axis in SI units (m/s^2). In practice, however, the measures are not very accurate due to the presence of gravitational and centripetal forces [14]. Thus, a mobile phone left to stand on a table presents a downward acceleration of about $9.82m/s^2$, the gravitational acceleration. It will be necessary a gravitational free fall toward the centre of the Earth to measure a value of zero in the three axes. Furthermore, the double integral that has to be solved for obtaining location from the acceleration value also increases the measurement error. Anyway, these sensors are handy for knowing the device's position relative to the floor with simple trigonometric calculations.

To test these devices a small application that simply takes the received measurements and save them in a file has been developed. Figure 2 depicts the values received while a user is walking along the Z axis with the mobile vertical to the floor (axis Y is perpendicular to the floor and axis X is on the side). As can be seen, there is a regular pattern of about a footstep per second crests in axis Y (continuous changes of about $4m/s^2$): acceleration progressively increases each time the user raises his foot to start a new step, and it falls when the foot reaches the floor, before starting a new step and so on. The lateral movement enclosed to each footstep can also be observed, but more complex movements would be hard to recognise, hence the difficulty of computing displacements using acceleration values. Broadly speaking, the accelerometers we have tested measure a lot of noise and therefore don't seem reliable enough for a real time application.

A digital compass or magnetometer is a device that measures the strength and direction of the magnetic fields in its environment. Due to the magnetic field of the Earth, a compass is usually employed as an orientation device since it points out the Earth's magnetic north. A smartphone usually incorporates a chip that integrates three compasses arranged to detect magnetic fields in the three spatial axes [15]. Thus, location can be obtained independently of the position of the device. Data are presented again as a three-component vector, with the rotation value for each axis. The first component of the vector is the rotation measurement usually employed, i.e., rotation with regard to z axis. This value is 0° when the device is oriented towards north direction and the value increases clockwise up to 360°, north again. The other two components, rotation regarding the other axes, are 0° when the device is lying face up. Figure 3 shows the results obtained by a test consisting of making an abrupt 90° turn, almost instantaneous, just before returning to the initial position by means of a slighter turn, during about 3 seconds. As can be observed in the figure, the compass is too slow in measuring the new position after the first sudden movement, what introduces wrong values during a short period. However, it behaves really well in the presence of slight movements, with accurate values and very little noise.

Therefore, to track the direction in which a smartphone moves with Android is recommended to take together data from the accelerometers and the compasses. By previously setting a default position for the device when the application starts we get enough accuracy, since measures of smooth changes in the local environment are quite precise.

GPS is a space-based global navigation satellite system that provides reliable location through an infrastructure comprising a network of satellites. This system can be used

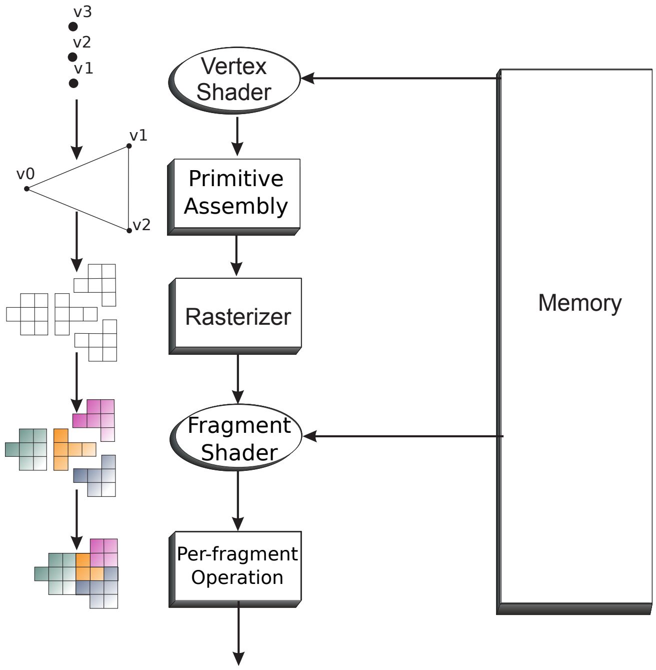


Figure 4: OpenGL ES 2.0 pipeline.

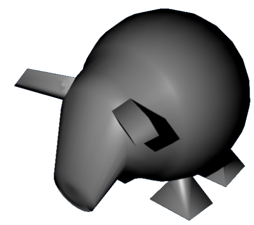


Figure 5: Test model for OpenGL ES.

Table VII: Performance (fps) of OpenGL ES in Android v2.1 and v2.2.

Points	GeeksPhone				Milestone				Galaxy S			
	C1	C2	C3	C4	C1	C2	C3	C4	C1	C2	C3	C4
3 K	35	35	35	33	30	30	30	30	55	55	55	55
9 K	18	19	19	15	30	29	29	28	55	55	55	55
15 K	12	10	10	10	29	26	26	25	55	55	55	55
30 K	8	-	-	-	25	22	22	19	55	55	55	55
75 K	-	-	-	-	18	15	15	12	55	53	53	50
100 K	-	-	-	-	-	-	-	-	55	44	44	41

all around the world whenever there is an enough number of visible satellites. Otherwise, less accurate measurements are obtained or the device can even get out of network coverage, usual problem in the indoor locations. The values obtained by a GPS device points out its current position in the globe with a few meters of precision, about 10 meters outdoor. Besides, it does not provide reliable information about the direction or inclination of the device and data is obtained with a delay of about 1 and 2 seconds. All this makes difficult to realistically locate and move synthetic objects that are close to the device.

Nowadays, an alternative method to GPS is network-based tracking by using the service provider's network infrastructure to identify the location of the device. Although this technique has less accuracy than GPS, it has the advantages of a shorter initialisation time and an important reduction in power consumption, since only the telephone signal is used. Additionally, it can provide better results in indoor environments than GPS. Anyway, both the two methods are compatible as they are not mutually exclusive.

C. Android and synthetic graphics

OpenGL ES [16] is the API for producing computer graphics in Android. It is a subset of the standard OpenGL designed for embedded devices, so it removes some redundant API calls and simplifies other elements to make it run efficiently on the low-power GPUs of handheld devices. Figure 4 depicts the rendering pipeline in OpenGL ES 2.0, with programmable hardware. Our codes are based on OpenGL ES 1.1, with the same pipeline but with configurable fixed function hardware instead of programmable shaders. A set of tests were carried out on the devices presented in Table I to analyse the performance of graphic synthesis in Android.

The first test focused on measuring performance as the number of primitives to render increases. The experimental results obtained for a scene with the model of Figure 5 replicated multiple times are shown in the column C1 of Table VII, in this case using the smartphones with Android v2.x: *GeeksPhone*, *Milestone* and *Galaxy S*. In view of these results, it is clear that the performance gets worse as the number of polygons increases except for *Galaxy S*, device in which we perceive a serious performance loss starting from 300K points. Non relevant values were not included in the table.

Column C2 of Table VII shows the results after adding a texture to the model of Figure 5. This definitely improves the visual aspect of the virtual objects with a minimum loss of efficiency, up to a 17% for a model of 75000 points in our *Milestone*. Column C3 depicts the results when including transparency effects. This hardly has influence on performance comparing to the synthesis with textures. In column C4 the results are obtained after applying illumination to the models. The performance decreases now a 24% in *Milestone* for a scene with 30K points. Obviously, this loss of performance is due to the additional computation required to get the colour of each pixel in the scene. Furthermore, it is necessary to define the light sources in the scene, setting its position, type, colour and intensity, in addition to provide each vertex of the models with a normal vector. As can be observed, the fall of performance in *Galaxy S* is only noticeable for models with a certain complexity (100K points).

Table VIII: Performance (fps) of OpenGL ES in Android v4.0 (also Android v2.3 in Galaxy S2).

Points	Sensation				Galaxy S2							
	C1	C2	C3	C4	C1		C2		C3		C4	
					2.3	4.0	2.3	4.0	2.3	4.0	2.3	4.0
30 K	60	60	60	60	60	60	60	60	60	60	60	60
75 K	56	55	55	54	60	60	60	60	60	60	46	52
100 K	54	53	53	50	59	59	60	60	60	57	33	37
150 K	49	48	49	24	56	58	60	60	60	60	23	26
180 K	43	40	37	18	50	55	49	50	52	56	20	22
210 K	37	32	33	14	45	48	43	44	41	46	17	19
240 K	31	26	26	12	40	43	38	39	37	39	14	17
270 K	25	21	23	10	36	38	32	35	35	37	13	15
300 K	21	17	16	9	31	35	30	32	31	33	11	13

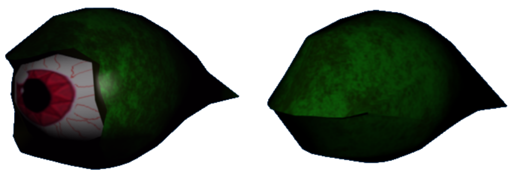


Figure 6: Morphing animation: starting state on the left and final state on the right.

Table IX: Frame rate comparison of static (S) and animated (A) models in the scene with Android v2.1 and v2.2.

Points	*GeeksPhone*		*Milestone*		*Galaxy S*	
	S	A	S	A	S	A
800	40	21	30	30	55	55
1.6 K	32	14	30	25	55	55
2.4 K	27	10	30	18	55	55
4 K	21	6	30	10	55	51
8 K	-	-	27	5	55	29
12 K	-	-	-	-	55	20
16 K	-	-	-	-	55	15

Table X: Frame rate comparison of static (S) and animated (A) models in the scene with Android v4.0 (also Android v2.3 in Galaxy S2).

Points	*Sensation*		*Galaxy S2*			
	S	A	S		A	
			2.3	4.0	2.3	4.0
30 K	60	60	60	60	60	60
75 K	60	53	60	60	51	60
100 K	59	39	60	60	36	60
150 K	57	28	60	60	24	46
180 K	55	25	55	60	19	39
210 K	54	21	47	60	17	33
240 K	54	19	42	60	14	30
270 K	47	17	41	53	12	27
300 K	44	14	37	48	11	23

Table VIII has the results for the smartphones with Android v4.0 used in our work: *Sensation* and *Galaxy S2*. *Sensation* keeps a good performance up to 180-210 K points, though the frame rate with illumination (column D) dramatically drops for more than 150 K points. *Galaxy S2* obtains very similar results to *Galaxy S*, keeping a good performance even with 300 K points. Again, working with illuminated models makes the performance drop, surprisingly resulting in poorer frame rates than *Sensation* for 75-100K points. The table also has the results for Android v2.3 in the Galaxy S2 model, what shows the noticeable improvement introduced with Android V4.0, above all when the number of primitives to be rendered increases.

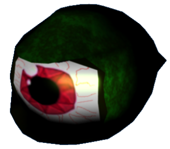
As regards animation, among all the different methods we have analysed the inclusion of morphing [17]. This technique gets a smooth transition between two models, using interpolation to compute the intermediate versions of the models. Since a new position for each point in the model has to be calculated for each frame, this kind of methods have a high computational cost. The model in Figure 6 (around 800 points and 300 polygons) has been used to test the performance of this kind of animation together with the application of textures and illumination in our target devices. The frame rates obtained for different scenes with this model on the smartphones with Android v2.x are shown in Table IX (only the most interesting results were measured). It can be observed that performance falls off dramatically except for low-complexity scenes (8K in the case of *Galaxy S*). These results are greatly improved in the new devices with Android v4.0, as shown in Table X. Specifically, the rendering of animated models has the most remarkable improvement, above all in the *Galaxy S2*, that exhibits a great performance up to 150 K points and keeps good frame rates with 240 K points in the scene. Again, the table allow us to compare the results accomplished by Android v2.3 and v4.0 when running on the same hardware (Galaxy S2). The improvements introduced in Android v4.0 lead to an increase in performance when the number of primitives to be rendered is greater than 150K.

D. Discussion

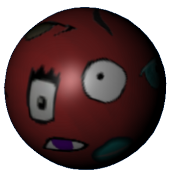
An important deficiency in the image processing capabilities of the platform has arisen, mainly in terms of image capture latency (a minimum of 0.246 seconds in high-end smartphones). The main augmented reality applications of other platforms use the information obtained after a complex analysis of the images captured by the camera as the main source of information for positioning the synthetic objects in the scene. In view of the results of our analysis, this kind of applications are currently not possible at all in the Android devices we have tested.

On the other hand, multiple conclusions can be extracted from the analysis carried out using the Android positioning sensors. First of all, regarding the use of the built-in locating and tracking sensors, the accelerometers and the compass provide results relatively reliable with no important errors.

(a) *BatGhost*



(b) *HulkGhost*



(c) *EmGhost*

(d) *SuperGhost*

Figure 7: Three-dimensional models.

However, GPS gives an excessive error in the measure to be used in the kind of AR indoor application we propose in this work.

Lastly, we have detected restrictions in size and complexity of the models to be rendered. From the results we can deduce that the graphic hardware is powerful enough to render non-excessively complex models with textures and illumination. Therefore, in the game we propose in the next section as an example of AR application, all the render capabilities we have analysed have been applied, but limiting the complexity of the model in order to get real time rendering.

III. An AR game in an Android platform

To exploit the different aspects we have studied in our analysis we have developed a simple AR game. In this game each real-time image obtained by the camera is analysed and it determines the apparition of 'enemies' that the user/player must hunt down. Thus, we have implemented a simple system of events based on object colours and the different enemies are drawn when a certain event is triggered. Therefore, in accordance with this game idea, the main requirement is to render virtual elements (the 'enemies') on the live image stream from the camera. These synthetic characters have to look as if they really were in the real world so they must behave properly with camera movements.

There are 4 different enemies in the game, each one of them with specific reactions and movements: *BatGhost* (Figure 7a), has been designed as an example of animation by parts, with its wings moving independently to provide a sense of flapping, *HulkGhost* (Figure 7b) with its blinking eye is an example of animation using morphing techniques, *EmGhost* (Figure 7c) was designed to have an enemy with bouncing capabilities, that could jump over the player, and *SuperGhost* (Figure 7d), that moves around the player while approaching to him. Whenever an enemy is hunted, the player earns points and extra shoots as a reward. Otherwise, if the player is hit by the enemy a life is lost. When the player loses its last life the game ends. Figure 8 shows some screenshots of the game: an enemy appears when the associated colour is detected through the camera (Figure 8a), the settings menu of the application (Figure 8b) and the colour calibration process (Figure 8c). This calibration is indispensable to get a right colour-based event triggering with different devices.

When it comes to rendering the different elements through OpenGL ES calls, the operating system itself executes these calls in a different thread, allowing a decoupled execution [18]. Furthermore, the reuse of memory is a constant issue in our implementation, preventing the number of memory allocations as far as possible.

Table XI: Frame rates of the AR game.

Test		*Milestone* ImPr	*Milestone* Syn	*GeeksPhone* ImPr	*GeeksPhone* Syn	*Galaxy S* ImPr	*Galaxy S* Syn
Static		3.25	15	2.75	8	4.10	35
	ImPr deact.		30		21		44
Anim.		2.75	8	2.50	3	3.60	23
	ImPr deact.		28		17		41

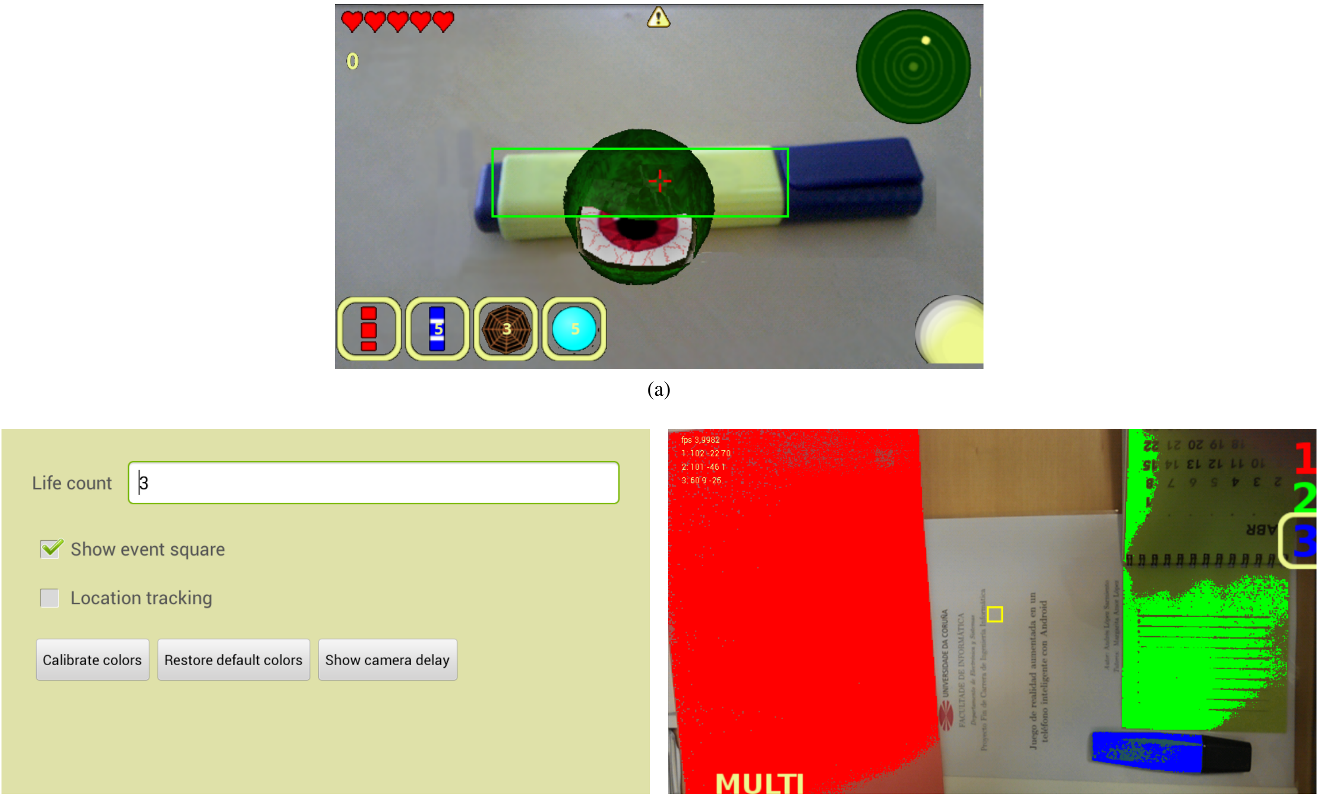
IV. Experimental results

This section presents the performance achieved by our AR application. The resulting frame rates are shown in Table XI. The different columns show the frames per second for image processing (ImPr) and image synthesis (Syn) in each device. The results in rows 2 and 4 (ImPr deact.) are obtained by deactivating the image processing task once an event is triggered, as described below.

In *Motorola Milestone* the image processing rate ranges from 3.25 fps with no visible enemy to 2.75 fps when an animated (morphing) enemy appears. Besides, the image synthesis rate falls down from 15 fps to 8 fps with only an animated model in the scene.

The performance is slightly worse in *GeeksPhone One*, with a peak of 2.75 fps for image processing. As can be seen, the main performance loss is mostly noticeable in the graphic synthesis. While the stream of images obtained from the camera is being processed, the performance values of the graphic synthesis are lower than the ones for *Motorola Milestone* in about 50%.

In the case of *Galaxy S* we have obtained better results, with a rate of image processing ranging from 3.6 fps to 4.1 fps along with a rate of synthesis of 35 fps for static models and 23 fps for animated, aspect in which the improvement is more appreciable.



(a) (b) (c)

Figure 8: Game screenshots: (a) Event detection and triggering (b) Settings menu (c) Color calibration.

On the other hand, the performance loss in the processing of the image has increased the delay in obtaining new images from the camera, reaching now about 1 second in our application.

As commented, once an enemy is discovered it does not keep still, it moves around the environment. To increase the frame rate and achieve a good response and fluid feeling we have stopped the image processing task while the enemy remains active in the screen. This restricts the appearance of multiple simultaneous enemies, but allow us to get an outstanding improvement in the rendering, reaching about 30 fps in *Milestone*, 21 fps in *GeeksPhone* and 44 fps in *Galaxy S*, a performance high enough to achieve an acceptable fluidity in an AR game.

V. CONCLUSIONS

In this work we present a study of the capabilities of current smartphones in the context of AR applications. Thus, to test the feasibility of this kind of applications we start checking the main constraints in the obtaining of data from the device's camera. The maximum frame rate we can obtained is less than 15 fps in the best cases. Including additional processing, such as colour segmentation, does not have an appreciable impact in performance. The main limitation is the latency in the image capture, near to 0.25 seconds in the best case. However, looking at the evolution of this delay, from about 1 second in the oldest analysed devices, rounding 0.5 seconds in the mid-range phones and about 0.25 in the *Galaxy S2* with Android v4.0, it seems that this drawback will be solved in a near future. Besides, using native code and the NDK seems essential to achieve a good performance, as the existing software layers are probably introducing the main performance issues (drawback of being a multiplatform OS).

Another point in our study has been to analyse the locating and tracking features of these devices. From our tests we have concluded that to obtain the device orientation is relatively simple and reliable. Nevertheless, to guess the device displacement is really complicated. Calculating it using the values obtained by the accelerometers is not very reliable, due to the numerical errors in the computation of the double integration. Additionally, geolocation systems have a margin of error too high for our requirements, about 10 meters.

With regard to the rendering of synthetic images with the OpenGL ES library, we have tested the inclusion of textures, illumination and transparencies. The performance achieved in scenes with up to 15K points has been acceptable for a mid-range smartphone as *Motorola Milestone*. Adding models with morphing animation means a loss higher than

20% each time the number of points is doubled.

As a proof of concept, to show the possibilities within the AR field of the different smartphones we have analysed, an interactive AR video game has been implemented. The performance we have achieved in this application is 3.25 images obtained through the camera per second and 28 fps in the synthesis of graphics in a mid-high end smartphone as *Motorola Milestone*. The results are better in a more powerful device as *Samsung Galaxy S*, 4.1 processed images per second and 35 fps, and appreciably worse in a low-end device as *GeeksPhone One*, 2.75 processed images per second and only 8 fps.

Additionally, it should be pointed out the great influence that the presence of other running tasks (background and foreground, e.g., a chat) have in the performance of an AR application. In this sense, a further analysis of the task scheduling on Android would be interesting.

As future work, we plan to propose a set of benchmarks in order to identify graphics processor bottlenecks. This proposal aims to guide programmers to identify the benefits of potential optimisations. These benchmarks could be a useful tool to make design decisions in architecture improvements.

Specifically, this proposal is aimed at designing, implementing and testing a set of benchmarks to analyse the rendering capabilities of Bézier surfaces on an Android smartphone with the OpenGL ES API. We expect to describe several hand-tuned Bézier Surfaces rendering in real-time implementation on Android systems, identifying key graphics processor performance limitations, enhancements and tuning opportunities.

ACKNOWLEDGEMENTS

This work has been economically supported by Ministry of Education and Science of Spain under the contracts MEC TIN 2010-16735 and TIN 2009-07737 and also by the Galician Government under the contracts 'Consolidation of Competitive Research Groups, Xunta de Galicia ref. 2010/6', 08TIC001206PR, INCITE08PXIB105161PR, INCITE08PXIB262202PR, and CN2012/211, partially supported by FEDER funds.

REFERENCES

[1] A. L. Sarmiento, M. Amor, C. V. Regueiro, and E. J. Padrn, "An analysis of android smartphones as a platform for augmented reality games," in *Proceedings of the Fifth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2011)*, November 2011, pp. 71–76.

[2] A. Inc., "iOS Dev Center," http://developer.apple.com/devcenter/ios/index.action, last access: 20/12/2012.

[3] S. E. Co., "Bada developers," http://developer.bada.com/, last access: 20/12/2012.

[4] M. Gargenta, *Learning Android*. O'Reilly, 2011.

[5] D. Wagner and D. Schmalstieg, "Making augmented reality practical on mobile phones," *IEEE Computer Graphics and Applications*, vol. 29, no. 3, pp. 12–15, 2009.

[6] Layar, "Layar reality browser," http://www.layar.com, last access: 20/12/2012.

[7] T. Langlotz, C. Degendorfer, A. Mulloni, G. Schall, G. Reitmayr, and D. Schmalstieg, "Robust detection and tracking of annotations for outdoor augmented reality browsing," *Computer & Graphics*, vol. 35, no. 4, pp. 831–840, 2011.

[8] MADfirm, "Sky siege," http://madfirm.com, last access: 20/12/2012.

[9] Quest-Com, "Droidshooting," http://www.quest-com.co.jp, last access: 20/12/2012.

[10] H. Kato, "Artoolkit," http://www.hitl.washington.edu/artoolkit, Android port by A. Igarashi (2010), last access: 20/12/2012.

[11] N. SL., "Invizimals," http://www.invizimals.com, last access: 20/12/2012.

[12] S. Lee and J. W. Jeon, "Evaluating performance of android platform using native c for embedded systems," in *Proceedings of the International Conference on Control, Automation and Systems*, 2010, pp. 1160–1163.

[13] "Android Google Code. Issue 2794: Camera preview callback memory issue," http://code.google.com/p/android/issues/detail?id=2794, last access: 20/12/2012.

[14] R. Meike, "Location, location, location (accelerometer)," https://blogs.oracle.com/roger/entry/location_location_location_accelerometer, last access: 20/12/2012.

[15] "Asahi Kasei Microdevices. 3-axis electronic compass," http://www.asahi-kasei.co.jp/akm/en/index.html, last access: 20/12/2012.

[16] D. Astle and D. Durnil, *OpenGL ES Game Development*. Thomson Course Technology, 2004.

[17] T. Akenine-Möller, E. Haines, and N. Hoffman, *Real-Time Rendering*. A. K. Peters, 2008.

[18] C. Pruett, "Writing real time games for android," http://www.google.com/events/io/2009/sessions/WritingRealTimeGamesAndroid.html, May 2009, last access: 20/12/2012.